Informatik I – Klausurvorbereitung

Fabian Wenzelmann

Die Aufgaben, die hier gestellt sind, dienen der Wiederholung. Diese Aufgaben decken nicht zwangsläufig den gesamten Stoff der Vorlesung ab. Relevant für die Klausur ist nur die Vorlesung! Keine Gewähr für Korrektheit, Vollständigkeit oder was auch immer!

Aufgabe 1

(a) Beweisen Sie mittels vollständiger Induktion, dass für alle $n \in \mathbb{N} \setminus \{0\}$:

$$\sum_{i=1}^{n} \frac{i}{(i+1)!} = 1 - \frac{1}{(n+1)!}$$

(b) Die n-te Fibonacci-Zahl f_n ist folgendermaßen definiert:

$$f_n = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ f_{n-1} + f_{n-2} & \text{sonst} \end{cases}$$

Beweisen Sie mittels vollständiger Induktion, dass für alle $n \in \mathbb{N} \setminus \{0\}$ gilt:

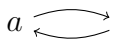
$$\sum_{i=1}^{2n} (-1)^{i-1} f_i = -f_{2n-1} + 1$$

Aufgabe 2

Betrachten Sie die Teilbarkeitsrelation $| \subseteq \mathbb{N} \times \mathbb{N}$ mit $a \mid b$ gdw. $\exists\, c \in \mathbb{N} : a \cdot c = b$ und $B := \{1, 2, 3, 4, 6, 9, 12, 18\}$. Bestimmen Sie alle maximalen, minimalen, größten und kleinsten Elemente von B bzgl. $|$.

Aufgabe 3

Gegeben sei die Relation $R \subseteq \{a, b\} \times \{a, b\}$. Als Graph:



Ist R

1. reflexiv?
2. symmetrisch?
3. transitiv?

Wenn nein: Begründen Sie Ihre Antwort!

Aufgabe 4

(a) Sie $R \subseteq A \times A$ eine Äquivalenzrelation. $R[a]$ bezeichnet die *Äquivalenzklasse* von $a \in A$ bzgl. der Relation R und ist definiert durch $R[a] := \{b \in A \mid aRb\}$. Also alle Elemente welche „äquivalent“ zu a sind.
Zeigen Sie: Für alle $a, b \in A$ gilt $R[a] = R[b]$ oder $R[a] \cap R[b] = \emptyset$.

(b) Sei $R \subseteq A \times A$ eine Halbordnung und $a \in A$ kleinstes Element von R bzgl. A. Zeigen Sie, dass

(i) a *eindeutig* kleinstes Element ist (es gibt kein weiteres kleinstes),

(ii) a minimal ist,

(iii) a das einzig minimale Element von R bzgl. A ist.

Aufgabe 5

Gegeben seien folgende Scheme-Prozeduren:

```
(: list-length ((list %a) -> natural))
(define list-length
  (lambda (xs)
    (if (empty? xs)
        0
        (+ 1 (list-length (rest xs))))))

(: f ((list integer) -> integer))
(define f
  (lambda (xs)
    (fold 0 + xs)))

(: double-with-minus ((list integer) -> (list integer)))
(define double-with-minus
  (lambda (xs)
    (if (empty? xs)
        empty
        (make-pair (first xs)
                   (make-pair (- (first xs))
                              (double-with-minus (rest xs)))))))
```

Beweisen Sie mittels *Listeninduktion*, dass für jede Liste ganzer Zahlen `l` (`(: l (list integer))`) gilt:

(a) `(list-length (double-with-minus l))` $= 2 \cdot$ `(list-length l)`

(b) `(f (double-with-minus l))` $= 0$

Hinweis: Sie dürfen dabei annehmen, dass die eingebauten Scheme-Prozeduren korrekt sind also das z. B. `(+ a b)` $= a + b$.

Aufgabe 6

(a) Schreiben Sie eine endrekursive Prozedur

```
(: sum (natural natural (natural -> number) -> number)
```

Dabei soll ein Aufruf `(sum k n f)` den folgenden Wert berechnen

$$\sum_{i=k}^{n} f(i)$$

Dabei sei per Definition

$$\sum_{i=k}^{n} f(i) := 0 \quad \text{für } k > n$$

(b) Schreiben Sie nun eine Prozedur `sum-lists` welche die gleichen Eingaben wie `sum` erwartet und den gleichen Wert berechnet ohne selbst eine rekursive Prozedur zu definieren. Sie dürfen die Prozedur `n-to-zero-list` (s. u.) verwenden.

```
; erstellt eine Liste natürlicher Zahlen mit allen Zahlen
   von n bis 0
(: n-to-zero-list (natural -> (list natural)))
(define n-to-zero-list
  (lambda (n)
    (if (zero? n)
        (list 0)
        (make-pair n
                   (n-to-zero-list (- n 1))))))
```

(c) Erklären Sie den Unterschied zwischen einer endrekursiven und einer nicht endrekursiven Prozedur anhand eines selbstgewählten Beispiels.

Aufgabe 7

Der Binomialkoeffizient $\binom{n}{k}$ ist die Anzahl der k-elementigen Teilmengen einer n-elementigen Menge und erfüllt folgende Rekursionsgleichung:

$$\binom{n}{k} = \begin{cases} 1 & \text{falls } k = 0 \\ 0 & \text{falls } n = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{sonst} \end{cases}$$

Schreiben Sie eine Scheme-Prozedur

```
(: binom-coeff (natural natural -> natural))
```

so dass ein Aufruf `(binom-coeff n k)` den Wert von $\binom{n}{k}$ berechnet.

Aufgabe 8

Die m-te *Bell-Zahl* B_m bezeichnet die Anzahl der Möglichkeiten, eine m-elementige Menge als Partition darzustellen, also eine m-elementige Menge als disjunkte Vereinigung von k nichtleeren Mengen darzustellen (ist nicht wichtig steht nur zur Vollständigkeit da ;)). Die Bell-Zahlen genügen der folgenden Rekursionsgleichung:

$$B_m = \begin{cases} 1 & \text{falls } m = 0 \\ \sum_{k=0}^{m-1} \binom{m-1}{k} B_k & \text{sonst} \end{cases}$$

Schreiben Sie eine Scheme Prozedur

```
(: bell-number (natural -> natural))
```

zur Berechnung der m-ten Bell-Zahl.
Einige Werte zur Korrektur: $B_5 = 52$, $B_{10} = 115975$.

Aufgabe 9

Eine *Priority Queue* ist eine Datenstruktur in der Elemente *geordnet* abgelegt werden was z. B. die Zeiten beim Suchen von Elementen reduziert. Eine Priority Queue soll dargestellt werden durch eine Liste von den Elementen in der Queue (`pq-elems`), und einer Vergleichsprozedur (`pq-<=`). Dabei sollen die Elemente der Liste stets folgende Eigenschaft haben: Wenn $\langle a_1, a_2, \ldots, a_n \rangle$ die Liste der Elemente ist, dann gilt: a_1 `pq-<=` a_2 `pq-<=` $\ldots$ `pq-<=` a_n. Eine Priority Queue wird in Scheme durch folgende Record-Definition repräsentiert:

```
; eine Priority-Queue (pq) ist ein Wert
; (make-pq pq-elems pq-<=)
```

```
; wobei pq-elems die Elemente der pq sind und pq-<= eine
   Ordnungsrelation auf den Elementen, die in der pq abgelegt
   werden und nach welcher die Elemente in pq-elems stets
   sortiert sein müssen - d.h. wenn (pq-<= a b) für zwei
   Elemente wahr wird, so steht a in der Schlange vor b. Wird
   ebenfalls (pq-<= b a) wahr, so spielt die Reihenfolge keine
   Rolle.
(define-record-procedures-parametric pq pq-sig make-pq pq?
  (pq-elems pq-<=))
(: make-pq ((list %a) (%a %a -> boolean) -> (pq-of %a)))

; Signaturkonstruktur für eine pq, welcher ein Argument erwartet
    und eine passende Signatur für eine pq erstellt
(define pq-of
  (lambda (t)
    (signature (pq-sig (list t) (t t -> boolean)))))
```

Schreiben Sie nun folgende Prozeduren, welche auf obigen Datentyp aufbauen:

(a) `(: initialize-pq ((%a %a -> boolean) -> (pq-of %a)))`, welche eine leere Priority Queue vom Typ %a erstellt

(b) `(: pq-insert (%a (pq-of %a) -> (pq-of %a)))`, welche ein neues Element in eine bereits existierende Priority Queue so einfügt, dass die Sortierung bzgl. der Vergleichsprozedur erhalten bleibt

(c) `(: sort ((list %a) (%a %a -> boolean) -> (list %a))`, welche eine Liste und eine Vergleichsprozedur als Eingabe erwartet und die Liste sortiert wieder ausgibt.

(d) eine Prozedur `(: is-sorted? ((%a %a -> boolean) (list %a) -> boolean))`, so dass `(is-sorted? p l) #t` gibt, wenn die Liste sortiert ist bzgl. `p` (Def. s. in Einleitungstext)

Aufgabe 10

(a) Schreiben Sie eine Python-Funktion `fib_rec` welche eine Zahl `n` als Argument nimmt und die n-te *Fibonacci-Zahl* rekursiv berechnet.

(b) Schreiben Sie eine Python-Funktion `fib_it` welche eine Zahl `n` als Argument nimmt und die n-te *Fibonacci-Zahl* iterativ berechnet d. h. ohne jede Art der Rekursion.